平成 n 年度 卒業論文

住井研究室の
ステキな論文クラスファイルの使用例

東北大学 工学部
情報知能システム総合学科

X0XX1234 ラムダ 小太郎

指導教員：住井 英二郎 教授

平成 n 年 1 月 1 日　23:00–23:30
電子情報システム・応物系 1 号館 2 階トイレ

要旨

ステキな論文の概要

謝辞

ステキな論文の謝辞

目次

第 1 章

序論

序論とか本論とか結論とか [1]

第 2 章

本論

2.1　ソースコード

ソースコード 2.1 は二分木を深さ優先探索して、ノードを列挙する関数である。

ソースコード 2.1　二分木のノードのリストアップ

```
1 type 'a bin_tree =
2   | Leaf of 'a
3   | Node of 'a bin_tree * 'a bin_tree
4
5 let rec listup_nodes = function
6   | Leaf x -> [x]
7   | Node (r, l) -> (listup_nodes r) @ (listup_nodes l)
```

ソースコードの書き方等については slide ブランチの slide.tex を参照されたし。

2.2　定理環境

第 3 章

使い方の例

3.1 BNF の書き方の例

本節では，BNF によるプログラミング言語の構文の書き方を紹介する．構文木の書き方は一つというわけではないので，幾つかのバリエーションを紹介する．どの方法が良いと思うかは，個人の好みに依るところなので，好きなものを使えば良いと思う．

まず，次の方法では，array 環境を使って，BNF を書いている．array 環境は数式環境中で表のようなものを書くときに使う．基本的に，table 環境と使い方は同じである．

$t ::=$		terms:
	x	variables
	$\lambda x.\ t$	lambda abstraction
	$t_1\ t_2$	application
	`true`	true
	`false`	false
	`if` t_1 `then` t_2 `else` t_3	if statement

他にも，次のように，align 環境を使っても，似たようなものを書くことができる．

$t ::=$		terms:
	x	variables
	$\lambda x.\ t$	lambda abstraction
	$t_1\ t_2$	application
	`true`	true
	`false`	false
	`if` t_1 `then` t_2 `else` t_3	if statement

array 環境を愚直に使う場合と比べて，式が中央揃えになるという点と，“variables” とかの説明が右端に来ている点が違う．説明は tag* マクロで出しており，これはもともと式番号を指定するためのものなので，若干使い方がおかしい気もするが，まぁ，いいだろう．自分の好みの方を使うと良いだろう．

BNF 全体を左揃えにしたいならば，次のように，flalign 環境を使うと良い．align 環境と違って，& を余分に 1 つ付ける必要がある，ということに注意して欲しい（詳しくはソースコードを見よ）．

$t ::=$	terms:
x	variables
$\lambda x.\ t$	lambda abstraction
$t_1\ t_2$	application
`true`	true
`false`	false
`if` t_1 `then` t_2 `else` t_3	if statement

3.2 導出木の書き方の例

導出木の書き方も色々あるが，ここでは，bussproofs.sty を使った方法を紹介する．導出木は，手書きでも書きにくいが，LaTeX だから書きやすいというわけでもなく，そこそこの苦労は必要である．bussproofs.sty を除く多くの方法では，frac などをベースに「分数」で導出木を書く．bussproofs.sty はこれらとは全く異なるインタフェースであり，（導出木を要素とする）スタックをイメージすると動作が解りやすい．よく使うマクロは次の通り．

- `\AxiomC{...}`：Axiom を push する（導出木では葉に相当）
- `\UnaryInfC{...}`：スタックから部分導出木（仮定）を 1 つ pop して，それを新たに作ったノード（結論）の子供にすることで，新たな部分導出木を作成し，push する．
- `\BinaryInfC{...}`：スタックから部分導出木（仮定）を 2 つ pop して，`\UnaryInfC` と同様の動作を行う．
- `\TrinaryInfC{...}`：スタックから部分導出木（仮定）を 3 つ pop して，`\UnaryInfC` と同様の動作を行う．

実際の使い方は以下の通り．

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ \text{T-Var}$$

$$\frac{\Gamma, x : T \vdash t : U}{\Gamma \vdash \lambda x.\ t : T \to U}\ \text{T-Abs}$$

$$\frac{\Gamma \vdash t_1 : T \to U \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1\ t_2 : U}\ \text{T-App}$$

$$\dfrac{\dfrac{\dfrac{}{x : \texttt{Bool} \to \texttt{Bool} \vdash \texttt{true} : \texttt{Bool}}\ \text{T-True}}{\vdash \lambda x.\ \texttt{true} : (\texttt{Bool} \to \texttt{Bool}) \to \texttt{Bool}}\ \text{T-Abs} \qquad \dfrac{\dfrac{y : \texttt{Bool} \in y : \texttt{Bool}}{y : \texttt{Bool} \vdash y : \texttt{Bool}}\ \text{T-Var}}{\vdash \lambda y.\ y : \texttt{Bool} \to \texttt{Bool}}\ \text{T-Abs}}{\vdash (\lambda x.\ \texttt{true})\ (\lambda y.\ y) : \texttt{Bool}}\ \text{T-App}$$

3.3　定理環境

この論文クラスファイルでは，デフォルトで以下の定理環境を提供している．

定理 3.1 (定理のタイトル) 定理の内容

補題 3.2 (補題のタイトル) 補題の内容

系 3.3 (系のタイトル) 系の内容

命題 3.4 (命題のタイトル) 命題の内容

定義 3.5 (定義のタイトル) 定義の内容

例 3.6 (例のタイトル) 例の内容

仮定 3.7 (仮定のタイトル) 仮定の内容

公理 3.8 (公理のタイトル) 公理の内容

証明 3.9 (証明のタイトル) 証明の内容 □

証明 (証明のタイトル) 証明の内容（番号なしの証明環境．証明を `\ref` で参照する必要がないなら，こっちを使うほうが自然かも） □

3.3.1　定理環境の使い方の例

補題 3.10 論文の中で最重要とは言えないような性質・命題は補題 (lamma) にする．補題や定理から直ちに導けるような軽い命題は系 (corollary) にする（細かい使い分けは人による）．

証明　`proof*` のように，アスタリスク付きの環境では，番号が付かない．

定理 3.11 提案手法の最も重要な性質や命題は，定理 (theorem) として書く．読者の心をくすぐる興味深いステートメントを書こう．

証明　定理 3.11 の華麗な証明．そのあまりにも美しい証明に，読者の目は釘付けだ！

Case 1. 自明
Case 2. 補題 3.10 から直ちに導ける．
Case 3. 言うまでもない．目を瞑れば証明が見えてくる．

Case 4. あんまり自明じゃない

(i) 自明じゃないと思ったけど，やっぱり自明だった

(ii) ほらね，こんなに簡単

第 4 章

結論

参考文献

[1] Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, 2002.